VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Rozšíření Vektorového animátoru – animační část

Extension of Vector Animator – Animation Part

2013 **Zdeněk Gold**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Zdeněk Gold**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Rozšíření Vektorového animátoru - animační část
Extension of Vector Animator - Animation Part

Zásady pro vypracování:

Cílem práce je rozšířit stávající projekt (jazyk Java) Vektorového animátoru o rozsáhlejší možnosti tvorby animace objektů (zmizení, pohyb po křivce, ...), jejich zbrazovaní na časové ose a editace vlastností animace. Další částí je vylepšení grafického uživatelského rozhraní o zobrazení časové osy animace a posuny v této časové ose.
Program bude rozšířen o:
1. Vkládání nových typů animace (zmizení, objevení, pohyb po křivce, ...).
2. Možnost editace vlastností vytvořených animací.
3. Zobrazení časové osy animace a s vyznačením jednotlivých objektů.
4. Posun zobrazení na konkrétní bod časové osy.
5. Možnost vypnutí a zapnutí zobrazení pomocných informací o animaci jednotlivých objektů (zobrazení cesty pohybu objektu, ...).
Práce musí obsahovat:
1. Popis řešení pomocí jazyka UML.
2. Implementaci popsané funkcionality.
3. Programátorskou a uživatelskou dokumentaci.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 2. 5. 2013

Zdeněk Šolc

podpis studenta

Poděkování

Rád bych poděkoval Ing. Davidu Ježkovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské práce a za vstřícnost. Rovněž bych rád poděkoval Veronice Machalové, za úpravu gramatických chyb v práci a její pomoc při strukturování textu.

Abstrakt

Práce si klade za cíl vytvořit jednoduchou, uživatelsky přívětivou aplikaci, která bude umět tvořit animace, pro podporu ve výuce. Dílo navazuje na předchozí vypracování Ing. Karlem Šutou, který zpracovával stejné téma a stanovil tak základ, který rozšiřuji. V první části práce se analyzuje funkcionalita existujících nástrojů pro tvorbu animací, vyzdvihují se přednosti, a je vytvořen návrh pro model animačního procesu. V části realizace se za pomocí UML diagramů popisuje řešení návrhu a popisuje se přidaná funkcionalita. Na závěr jsou vyjmenovány možné způsoby rozšíření aplikace o další funkce.

Klíčová slova

Java, animace, animační proces, třída, balík, instance, akcelerátor, panel

Abstract

The work aims to create a simple, user-friendly application that will deal with the creation of animations to support the teaching process. The work builds on previous development by Ing. Karel Šuta, who worked on the same topic and set the foundation which I extend it. The first part analyzes the functionality of existing tools for creating animations. Advantages are listed and created a proposal for animation process model. In the implementation part with the help of UML diagrams is described solution design and added functionality. In the end possible ways to extend the application of additional functions are listed.

Key words

Java, animation, animation process, class, package, instance, the accelerator, panel

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
XML	Extensible Markup Language	Značkovací jazyk
GUI	Graphical User Interface	Grafické uživatelské rozhraní
UML	Unified Modeling Language	Unifikovaný modelovací jazyk
FPS	Frame Per Second	Počet snímků za sekundu
GC	Garbage Collector	Nástroj k čištění paměti v Jave

Obsah

1 Úvod

Cíl této bakalářské práce spočívá v rozšíření návrhu nástroje pro tvorbu animací, sloužící jako opora při tvorbě prezentací a k výkladům látky, zejména tam, kde je třeba nasměrovat představu o funkčnosti problematiky.

Z předešlého vypracování diplomové práce Ing. Karla Šuta, jsem vypsal všechny nedostatky Vektorového animátoru a stanovil cíle, které je třeba dotvořit, aby byla aplikace funkční a jednodušší pro uživatele. Tam, kde je to možné, zůstala kostra architektury s použitím návrhových vzorů zachována a byla pouze obohacena o nové třídy, pro docílení aplikačně funkčních požadavků. Také jsem analyzoval existující aplikace na trhu a inspiroval se podobnou funkcionalitou.

V části pro popis implementace je za pomocí UML diagramů popsáno řešení rozšíření funkcí s návrhem možného budoucího rozšíření či změny.

Práce je vypracována v součinnosti s bakalářskou prácí studenta Martina Golda, který zpracovává druhou část rozšíření, a to rozšíření aplikace o nové tvary, možnosti nastavení těchto tvarů a návrhu celkového vzhledu aplikace. Součinnost byla prováděna za pomocí verzovacího nástroje a repozitáře obsahujícího projekt.

2 Použité nástroje a technologie

K vypracování návrhu a implementace rozšiřující funkčnosti jsem se opíral o existující pomocné nástroje, které mi umožnily korektní a rychlou tvorbu některých problematických částí. Jedná se o seznam těchto nástrojů.

- **Xuggler [1]**

  Podpůrná knihovna pro Javu, k práci se zvukem i videem. Byla použita pro export animační scény jako video soubor.

- **Visual Paradigm for UML [2]**

  Nástroj pro tvorbu vzorových UML diagramů. Obsahuje nepřebernou škálu tvorby diagramů: třídní, sekvenční, stavový, business, kontextový, apod. Má přehledně zpracované uživatelské rozhraní s projektovým plátnem, na které se za pomocí myši tvoří diagramy. Výhodou je, že obsahuje možnost českého prostředí.

- **Trotoise SVN [3]**

  Verzovací nástroj, který byl použitý k ukládání jednotlivých verzí aplikace, při její tvorbě.

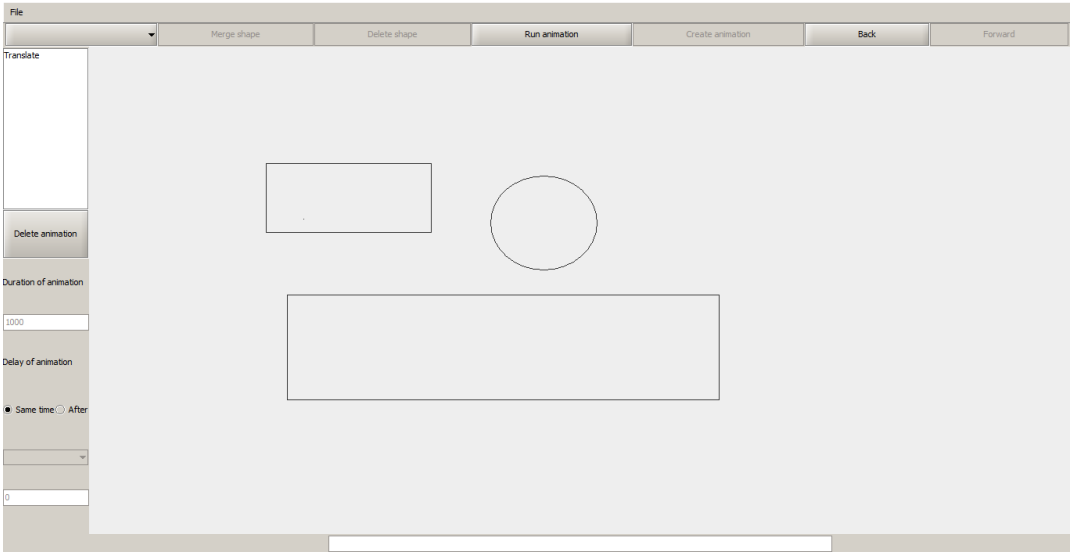
3 Analýza a návrh vypracování

Kapitola popisuje průběh návrhu rozšíření o další animační prostředky, a v případě rozsáhlejších změn v celém procesu výpočtu animací a jejich zobrazení, zdůvodňuje použitý přístup.

3.1 Analýza nedostatků předešlého zpracování

Předešlá verze aplikace se potýkala s nepříliš intuitivní tvorbou animace. Hlavní problém se objevil při tvorbě animace, kdy nebyla možné vidět náhled. Výsledek aplikování animace na objekt jsme si mohli překontrolovat až při samotném spuštění a zároveň se animovaný objekt při skončení celé animace nevrátil zpět do původního stavu.

Animace byly v tomto případě tvořeny na základě stisknutí, tažení a uvolnění tlačítka myši. Takovýto přístup není vhodný při definování animací pohybu po křivce, kde je třeba definovat více než jen dva body. Rovněž chybí jakákoliv dodatečná editace vytvořené animace. Pokud se při tvorbě dopustíme chyby a špatně definujeme animaci, je třeba celou animací smazat a začít s ní od začátku.



Obrázek 3 .1: *Původní verze aplikace Animator*

Slabinou aplikace je rovněž fakt, že animace lze skládat pouze s navázáním na jinou animaci a nelze si jednoduše vyvodit, kdy se která animace spustí a jak bude objekt transformován v určitém čase. To všechno bylo třeba změnit a vyplnit mezery funkčnosti předchozí verze.

3.2 Analýza funkčnosti dostupných aplikací

Na internetu lze nalézt hned několik aplikací, které se zabývají tvorbou animací vektorových obrázků. Celé spektrum těchto aplikací lze rozdělit na kategorii „free“ nástrojů, které jsou volně ke stažení s bezplatnou licencí a na kategorii programů, které obsahují větší množství efektů a nástrojů, ale za větší nebo menší finanční obnos.

Tabulka.3.1: Dostupné aplikace

Bezplatná licence	Synfig Studio, Stickman 5
Placená licence	SWiSH Max4, Microsoft PowerPoint 2007

Při analýze těchto aplikací jsem se zaměřil na to, jakým způsobem funguje tvorba animací, jak se reprezentuje jejich skládání, zda je proces tvorby a úpravy animací jednoduchý a časově nenáročný, jaké jsou nabízené efekty animací, zda se zobrazují pomocných informací animace (trajektorie, velikost rotace, …) a jestli je také možné vidět časové uspořádání animací na ose. To vše vedlo k určení cílů pro rozšíření aplikace.

3.2.1 Stickman 5

Program Stickman 5 je primárně uzpůsoben k tvorbě „skeletonů“ a k tvorbě klíčových snímků, zachycující stavy objektů. Animování pak funguje tak, že se dopočítává stav objektu mezi dvěma snímky.

Tato aplikace není vhodná pro rychlou tvorbu výukových prezentací, protože je pro první použití příliš složitá. V nástroji také chybí možnost tvorby primitivních objektů. Animace jsou velmi omezené, protože se stav objektů přepočítává mezi dvěma stavy. Tudíž by animace pohybu objektu po křivce vyžadovala ruční vložení snímků pro každý stav pohybu po křivce.

Výhody:

- Tvorba pohyblivých postav „skeletonů“
- Dobře implementovaná časová osa
- Export projektu do video souboru

Nevýhody:

- Chybějící animace (rotace, rozšíření, zmizení, pohyb po křivce)
- Chybějící možnost skládání animací

3.2.2 SWiSH Max 4

Program SWiSH Max 4 je komerčním nástrojem pro tvorbu multiplatformních prezentací pod platformou Adobe Flash. Aplikace má stejně jako v případě Stickman 5 přehledně zpracovanou časovou osu se zobrazením všech objektů a jejich animací. Animace se tvoří pomocí klíčových snímků, mezi kterými se vypočítává stav animace.

Pokud ovšem budeme potřebovat vidět náhled celé animace v reálném čase, budeme si muset do počítače nainstalovat software Adobe Flash Player. Bez něj se musíme spokojit pouze s náhledem v jednotlivých snímcích.

Výhody:

- Časová osa s vyznačením jednotlivých objektů na plátně
- Zobrazení pomocných informací o průběhu animování objektu (odkud, kam)
- Export projektu do video souboru

Nevýhody:

- Animace nelze kopírovat za sebe
- Nelze definovat nelineární průběh animací

3.2.3 Microsoft PowerPoint 2007

Tento komerční software, který patří do balíčku kancelářských programů Microsoft Office 2007, se zaměřuje především na tvorbu prezentací. Nicméně i přes to, že není primárně určen k tvorbě animací, poskytuje možnost vložení vlastní animace. Animace jsou aplikovatelné na všechny vložené objekty včetně písma. Existuje zde bohatý výčet funkcí: posunutí, rotace, rozšíření, změna barvy, písma, řezu, různé typy zobrazení, skrytí, pohyby po vlastních nejrůznějších trajektoriích a další. Aplikování potřebného efektu je také velmi jednoduché a rychle použitelné. Také se mi líbí, že se animace dají uspořádávat, skládat a dodatečně upravovat, aniž by se tím ovlivnil stav objektu.

Hlavní nevýhodou tohoto nástroje je skrytá časová osa, tudíž nemáme možnost náhledu animace v určeném čase. Editace animací probíhá skrze dialogová okna, což do značné míry zdržuje tvorbu výsledné scény. V neposlední řadě může být omezující i fakt, že z hlediska nastavení vlastností funkcí se musíme smířit s předdefinovanými hodnotami.

Výhody:

- Široká nabídka animací
- Náhled efektu animace před vložením
- Ručně zadaný pohyb po neomezeně dlouhé trajektorii
- Režim výsledného náhledu na celé obrazovce
- Definování počtu opakování, zpětné převinutí, zpoždění

Nevýhody:

- Skrytá časová osa
- Nelze nastavit parametry animace na konkrétní hodnoty

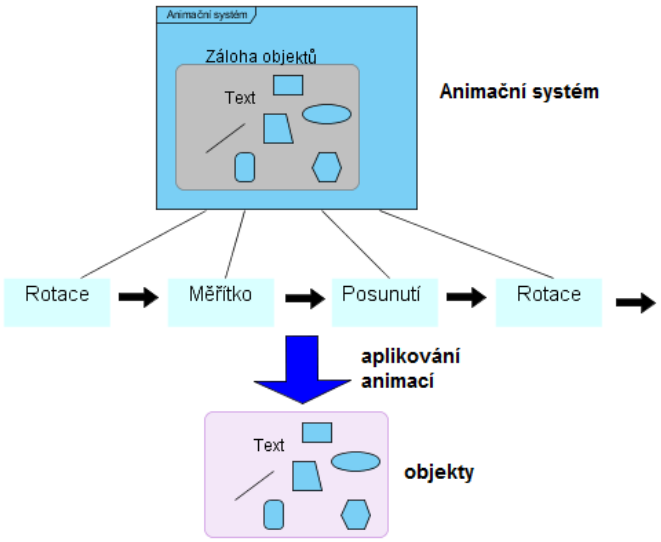
3.3 Analýza animačních modelů

Máme k dispozici dvě možnosti, jakým můžeme uvažovat o tvorbě animované prezentace a tvorbě animací vůbec. První možnosti je přístup v podobě ukládání pozic a vlastností objektů v dané časové pozici. Tento postup je v literatuře označován jako tvorba animací s pomocí **klíčových snímků**. Druhou možností je použití **funkční logiky** nad objekty (rotace, posunutí, změna měřítka, apod.) a při dosažení nastaveného času, tyto funkce aplikovat.

3.3.1 Animace řízené klíčovými snímky

Animace s využitím klíčových snímků můžeme nalézt například u programu Pivot Stickfigure Animator a jemu podobným programům. Animace je definována na základě rozvržení objektu a jeho vlastností v klíčových snímcích a v ostatních snímcích se pouze dopočítává stav objektu na základě předchozího a následného klíčového snímku. Výhodou je značná kontrola nad pohybem objektů. Chování animace si lze přesně definovat na základě dílčích změn. Příklad využití můžeme nalézt třeba při kompresi videa.

Výhodou je rychlost výpočtu pozice objektu mezi klíčovými snímky, protože se využívá dopočítávání pozice z předešlého či následujícího klíčového snímku, který už je uložený v paměti. Jak je patrné, je tento postup náročný na paměť. Avšak pokud se jedná o vektorovou grafiku, není paměťová náročnost zase takovým problémem. Nevýhodou je, že není možné definovat složitější pohyby objektů, jako je například rotace kolem bodu nebo volný pád.



Obrázek 3 .2: *Model animací řízenými funkční logikou*

3.3.2 Animace řízené animačními funkcemi

Animační model založený na animačních funkcích vychází z modelu animování s klíčovými snímky. Na místo uložení cílových a počátečních stavů objektů a dopočítávání pouze stavů mezi těmito klíčovými snímky, se ukládají samotné přechodové funkce animací a jejich pořadí vykonávání. Jde o množinu funkcí, které lze realizovat nad libovolným objektem. Může to být kupříkladu přemístění objektu na novou pozici, která je definovaná číslem nebo pozicí, rotací tvaru, rozšířením, zkosením, změnou barvy, apod. Představa takovéhoto modelu je zachycena na obrázku 3.2

Tento model ještě více odlehčuje paměťovou náročnost, protože je v paměti uložen pouze jeden stav objektu a další se dopočítává na základě animační funkce a na základě aktuálního stavu objektu. Mnohem vyšší je výpočetní náročnost. Ovšem s ohledem na rozsáhlost scény a množství použitých funkcí, nejde o tak markantní nárůst.

Přechod mezi snímky

Přechodem z jednoho časového bodu do jiného je třeba dopočítat nový stav objektu v tomto čase. U animačních systému reprezentovaných klíčovými snímky to je prosté. Vzal se nejbližší předešlý klíčový snímek a nejbližší následující klíčový snímek a na základě dvou stavů se dopočítal stav objektů kdekoliv mezi tím. U způsobu realizace animací jako funkcí se nám naskýtají dvě možnosti: výpočet stavu na základě aktuálního stavu (**relativní**), či přístup s využitím uloženého počátečního stavu (**absolutní**). Obě možnosti mají své pro i proti.

V případě relativního výpočtu je předností to, že pokud nepožadujeme výpočet mezi časově vzdálenými stavy, je výpočet velice rychlý, a tedy i vhodný pro zobrazení v reálném čase. Na druhou stranu je však tento systém zpracování náchylný na chyby a ztrátu přesnosti, způsobenou přímou závislostí toho, jak přesně je v počítači realizováno reálné číslo. Každým výpočtem se tato chyba inkrementuje a po čase začne být znát.

S využitím počátečního stavu se v každém čase animace aplikují od počátečního stavu všechny animační funkce v pořadí, jaké jim náleží. Výpočetní náročnost je tím větší, čím jsme blíže konci celé animace. Přesto je však výpočet velice přesný a nedochází k chybám.

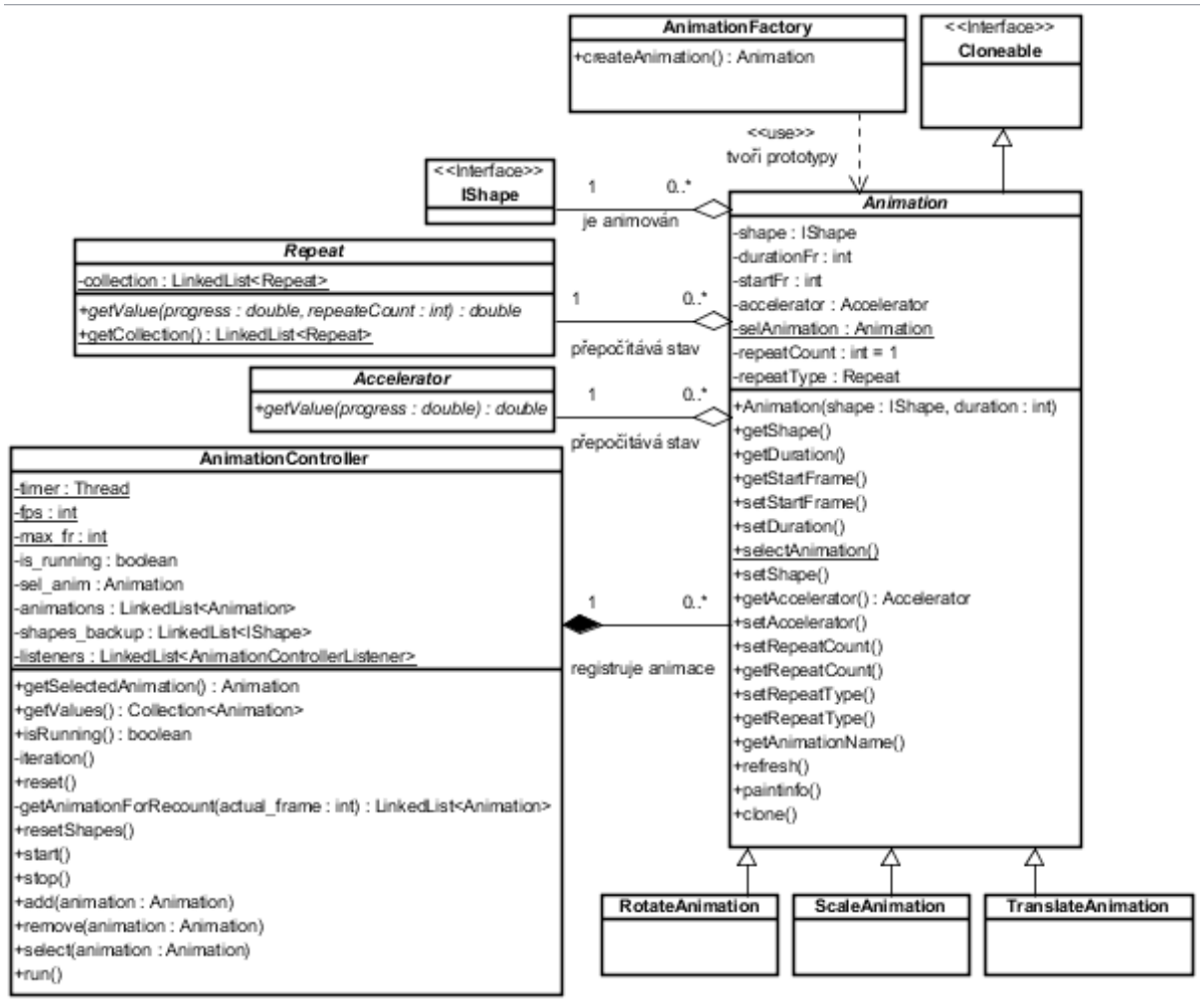
3.4 Volba animačního modelu

Pro implementaci animací v tomto programu jsem zvolil animační model s využitím animačních funkcí. Tato možnost nabízí elegantní způsob definování animace na základě předpisu funkce a ulehčuje tak práci s definováním klíčových snímků, protože například pro definování rotace kolem bodu by muselo být vytvořeno mnoho klíčových snímků, což by zahlcovalo paměť. S ohledem k nepřesnostem při práci počítače s reálnými čísly, bude výpočet realizován na základě počátečního stavu objektu. Přejde se tak problémům s přesností a chybovostí při přechodu z jednoho snímku na nový.

4 Implementace animací

4.1 Animační model

Pro implementaci funkčnosti animací jsem se na základě analýzy, které jsem se věnoval v předešlé kapitole, rozhodl pro model animací, řízený animačními funkcemi s počáteční zálohou objektu. Jelikož není předpokládáno, že bude třeba vytvořit rozsáhlé dlouhé scény s desítkami animací, nebude výpočetní náročnost překážkou k dosažení obnovovací frekvence při náhledu. Taktéž nebude docházet k inkrementaci chyby při výpočtech s reálným číslem. Důležitý je fakt, že se vždy vychází ze základního stavu objektu, jehož stav se pro animační požadavky naklonuje a uloží do třídy **AnimationController**. Struktura modelu je vyobrazena na obrázku 4.1.



Obrázek 4 .1: *Animační model*

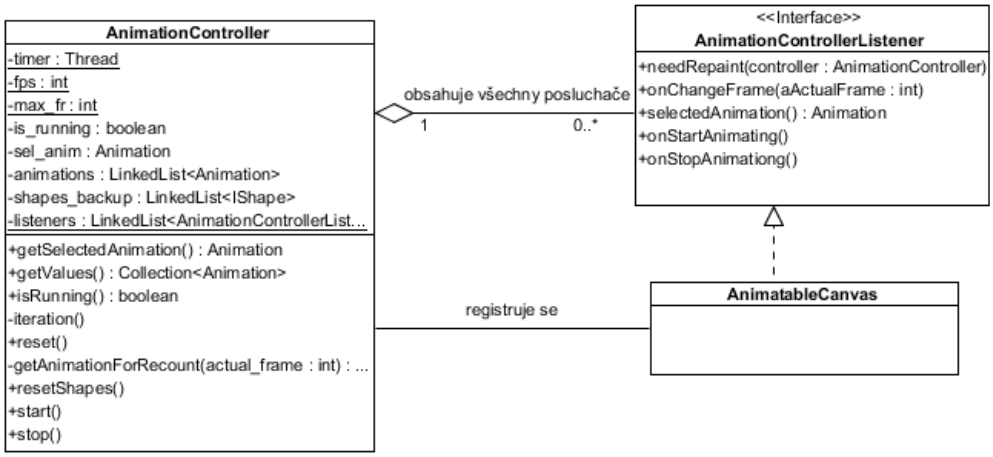
Animační model je tvořen třemi hlavními třídami: **AnimationController**, **Animation** a **AnimationFactory**. Třída **AnimationController** uchovává kolekci animačních instancí, které se mají při spuštění celé animace aplikovat na vybrané objekty. Poskytuje možnost spuštění a zastavení

animací v určitém čase a přechod na jiný čas. Všechny animační třídy musí dědit z abstraktní třídy **Animation** a obsahovat tak členské proměnné pro definování společných vlastností. Třída **Animation** reprezentuje rodičovskou třídu konkrétních typů animací. Má v sobě nadefinovaný algoritmus pro vykonání konkrétní změny s vybraným objektem a možnost nakonfigurování parametrů animace.

4.2 Třída AnimationController

Třída **AnimationController** je srdcem celé koncepce řízení animací v čase. Tato třída je tvořena metodami, jak pro spuštění a zastavení animací, tak i metodami a funkcemi, které nastavují objekt na požadovaný snímek nebo vracející kolekci animačních funkcí, které se vykonají do stanoveného času. Nejdůležitější je reprezentace registrace a zrušení animace. Po aplikování animace na objekt se vytvořená instance třídy, dědící z abstraktní třídy **Animation,** zaregistruje přes metodu ***addAnimation*** a od této chvíle je spravována touto třídou.

Třída **AnimationController** spouští vlastní vlákno, které hlídá správné časování s obnovovací frekvencí (FPS) pro výpočet, která je implicitně nastavena na 25 snímků/s. V každém snímku je proveden jeden cyklus přepočítání objektů právě běžících animací metodou ***iteration***, a poté je rozeslána událost události o změně stavu objektu třídě **AnimatableCanvas**. Způsob komunikace mezi těmito třídami je reprezentován obrázkem 4.2 a je představuje návrhový vzor Observer [5].



Obrázek 4 .2: *Komunikace mezi třídou AnamatableCanvas a AnimationController*

K zajištění přesnosti ve výpočtech změn objektů se využívá zálohování stavu animovaného objektu při vložení animace a jeho využití při výpočtech změn tvaru. Zálohy jsou uloženy v kolekcích. Jde vlastně o reprezentaci animačního modelu realizovaného funkcemi s počáteční zálohou.

4.3 Třída Animation

Třída **Animation** slouží jako společný rodič všech nadefinovaných animačních tříd, takže poskytuje společnou funkčnost a prostředky konkrétním třídám animací. Aby bylo možné všechny animace vytvářet na základě jejich prototypů, implementuje rozhraní Cloneable, pro překrytí metody clone. Třída je abstraktní, jelikož definuje operace, které je třeba implementovat, ale samotnou implementaci nechává až na potomcích. Konkrétní případem je metoda ***refresh***, starající se o výpočet

aktuálního stavu animovaného objektu v čase. Je jasné, že implementace závisí na konkrétním algoritmu konkrétní animační funkce.

Mezi členské proměnné této třídy patří proměnná *shape*, které odkazuje na animovaný objekt, *startFr* a *durationFr*, určující kdy a na jak dlouho má být animace aktivní. Pro docílení nelineární rychlosti průběhu celé animace, která je typická pro většinu animačních nástrojů, byla do animačního modelu přidána třída **Accelerator**. V případě několikanásobného opakování jedné animace byla implementována třída **RepeatType**.

4.4 Třída Accelerator

Třída **Accelerator** umožňuje definovat, jak se má měnit rychlost změny objektu. Představíme-li si např. přesunutí objektu z bodu A do bodu B, můžeme předpokládat, že bude pohyb probíhat s konstantní rychlostí. V některých případech se nám ale hodí pohyb, kdy se zpomaluje nebo zrychluje pohyb objektu.

Třída **Accelerator** používá pro definici rychlosti průběhu jedinou abstraktní metodu ***getValue*** s parametrem, představujícím stav, v jakém se animace nachází. Jako výsledek vrací metoda novou hodnotu stavu animace vzhledem k typu „akcelerátoru“ a jeho předpisu. Stav animace může být libovolná hodnota od 0 po 1. Na příkladě zdrojového kódu si demonstrujeme funkci akcelerátoru.

```
public double getValue(double progress) {
      return progress * progress;
}
```

Hned na první pohled zjistíme, že jde o předpis kvadratické funkce. Taková funkce se například hodí při definování zrychlujícího pohybu.

V případě definování vlastního „akcelerátoru“, se musíme omezit pouze na předpisy funkcí, které jsou rostoucí pro celý definiční obor <0,1> a které mají obor hodnot v rozmezí <0,1>.

4.5 Třída Repeat

Třída **Repeat** je abstraktní třída, sloužící k možnosti měnit průběh animace v závislosti na počtu opakování. Máme na výběr ze dvou využitelných možností: „vždy od začátku“, kdy po příchodu animace do koncového stavu, přejde okamžitě do stavu počátečního, nebo „tam a zpět“, která v okamžiku přejití do koncového stavu změní směr a plynule přejde do stavu počátečního.

Třída spravuje kolekci všech potomků této třídy, kteří se automaticky zaregistrují do této kolekce přes konstruktor rodiče. Pokud tedy kdekoliv v kódu vytvoříme instanci potomka této třídy, přidá se tato instance i do kolekce. O prvotní inicializaci všech potomků se stará konstruktor třídy **AnimationController**. Potomci třídy **Repeat** se vyskytují v balíku *dialog.animation.repeat*.

4.6 Třída AnimationFactory

Třída **AnimationFactory** stojí za implementací dynamického načtení všech animačních tříd a tvorbě množiny poskytovaných animací. Mimo jiné je určena k získání instance prototypu animace.

V aplikačním kódu je tato třída zodpovědná za tvorbu animací klonováním prototypu jedné z načtených animačních instancí.

4.7 Použité animace

V této sekci následuje soupis implementovaných animací s krátkým popisem funkčnosti a představení jejich nastavitelných parametrů. Každá animační funkce je reprezentována třídou, která dědí z abstraktní třídy **Animation**. K docílení funkčnosti konkrétní animační třídy, může animační třída využívat funkčnosti rodiče.

Většina animací využívá k výslednému efektu přesouvání bodů objektu. K tomuto účelu se skvěle hodí využití grafických transformací (otočení, změna měřítka, posunutí a zkosení). Java nabízí možnost vytvoření a skládání těchto transformačních matic za pomocí třídy s názvem **AffineTransoform**. Proces aplikování transformační matice na každý bod objektu je pak mnohem rychlejší, než výpočet pro každý bod zvlášť.

Jelikož může být animovaný objekt vytvořen jako kompozit a složen z více elementárních objektů nebo dalších kompozitů, bylo třeba na to při tvorbě funkcí pro animování brát ohled. Z implementace třídy **CompositeShape** (kompozitu) je zřejmé, že metoda ***getPoints*** nám nevrátí všechny body všech objektů v kompozitu, ale jen krajní 4 body pro označení. Správným postupem je přetypování objektu na objekt třídy **CompositeShape** a volání metody ***getAllChildrens***, která vrátí všechny elementární objekty kompozitu. Až tehdy voláme metodu ***getPoints***, nad každým objektem.

Rotace

Animace rotace je reprezentována třídou **RotateAnimation**. Animace funguje tak, že otáčí body kolem vybraného středového bodu, kterým může být buď referenčním bodem objektu, nebo může objekt rotovat kolem vlastního bodu definovaného jako vzdálenost X a Y od referenčního bodu. Pozice referenčního bodu je v tomto případě přepočítávána na kartézský souřadný systém s orientací osy y směrem nahoru a osy x směrem doprava od počátku.

Velikost pootočení objektu je dána proměnnou *rotateValue*. Směr otáčení je v případě kladných hodnot této proměnné ve směru hodinových ručiček. V případě záporných zase naopak.

Přesunutí

Animace přesunutí je reprezentována třídou **TranslateAnimation**. Tato třída plní svou funkci tím, že animuje přesunutí objektu na plátně o definovanou relativní vzdálenost vzhledem k referenčnímu bodu objektu. Tato hodnota je ve třídě reprezentována dvojící proměnných: *move_x* a *move_y*, vyjadřující x-ové a y-ové přesunutí. Stejně jako u rotace je hodnota přepočítávaná tak, aby dolní levý roh plátna znamenal počátek souřadného systému.

Změna měřítka

Animace změny měřítka je reprezentována třídou **ScaleAnimation**. Funkce změna měřítka je příkladem aplikování zobrazení středové souměrnosti pro objekt s nulovým pootočením. V podstatě jde o změnu šířky a výšky objektu vzhledem k definovaným koeficientům. Ve třídě to jsou proměnné *x_scale* pro šířku a *y_scale* pro výšku. Výsledná velikost je pak přenásobením koeficientů šířkou a

výškou. Hodnoty větší než 1 způsobují zvětšení velikosti, zatímco menší naopak zmenšení. Hodnoty záporné způsobí zrcadlení objektu vzhledem k jeho x-ové nebo y-ové ose.

Za výpočtem změny měřítka stojí metoda třídy **R2_Transformation** s názvem ***scaleTransform***. Metoda provádí zobrazení bodů objektu za pomocí středové souměrnosti podle referenčního bodu objektu. Avšak aby byl objekt rozšiřován vzhledem ke svoji šířce a výšce, je ve výpočtu pootočen souřadný systém, v závislosti na otočení objektu.

```
void scaleTransform(IShape shapes, double scale_x, double scale_y) {
      AffineTransform translation = new AffineTransform();
      ShapePoint tempPoint =
      shapes.getRootShape().getReferencePoint().getNewInstance();
      double angle = Math.toRadians(shapes.getAngle());

      translation.translate(tempPoint.getX(), tempPoint.getY());
      translation.rotate(angle);
      translation.scale(scale_x, scale_y);
      translation.rotate(-angle);
      translation.translate(-tempPoint.getX(), -tempPoint.getY());
      Mediator.R2.applyAnimation(shapes, shapes, translation);
}
```

V ukázce kódu vidíme, že před aplikací „scale“, což je naše rozšíření, se objekt nejprve přesune svým referenčním bodem do počátku souřadného systému, potom se pootočí zpátky a až nakonec se rozšíří/zmenší.

Pohyb po křivce

Animace pohybu objektu po křivce je definována třídou **BezierCurveAnimation**. Křivka je reprezentována Bézierovou kubickou křivkou, což je Bézierová křivka třetího stupně, která má 4 body (počáteční a koncový bod a k tomu dva body řídící). Výhodou této křivky je možnost navázání více těchto křivek na sebe a jejich spojitost v koncovém bodě. Pohyb objektu po křivce je tedy realizován jako dílčí posunutí v čase t.

$$C(t) = \sum_{i=0}^{3} \binom{3}{i} t^i (1-t)^{3-i} P^i \tag{4.1.}$$

$$(t^3 \quad t^2 \quad t \quad 1)\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \tag{4.2.}$$

Objevení

Animace postupného zviditelnění objektu je reprezentována třídou **ShowAnimation**. Tato funkce aplikovaná na objekt postupně u barvy výplně, barvy ohraničení a druhé přechodové barvy snižuje hodnotu alfa kanálu, aby zprůhledňovala objekt. V případě složených objektů mění průhlednosti všech objektů, ze kterých se složený objekt skládá.

Zmizení

Animace postupného zmizení objektu je reprezentována třídou **HideAnimation**. Tato funkce aplikovaná na objekt postupně zvyšuje hodnotu alfa kanálu u všech barev objektu a objekt se tak stává viditelnějším, až do úplného zviditelnění. U složených objektů je tento algoritmus aplikován na všechny objekty, ze kterých je složen.

Změna barvy

Animace přechodu z aktuální barvy na zvolenou barvu je reprezentována třídou **ChangeFillColorAnimation**. Každý objekt má svoji základní barvu, kterou lze získat metodou ***get_fill_paint_1*** z rozhraní **IShape**. Při postupné změně barvy na zvolenou barvu se neděje nic jiného, než že se vytvoří nová barva výplně, která leží na pomyslné úsečce mezi body, reprezentující původní a výslednou barvu v barevném modelu RGB. Vypočtenou barvu aplikujeme na animovaný objekt voláním metody ***setFill_paint_1***.

Změna průhlednosti

Animace změny průhlednosti je zastoupena třídou **TransparencyAnimation**. Od předešlých animací zmizení nebo objevení, se liší pouze v tom, že lze, za pomocí parametru nastavitelného pomocí metody ***setTransparencyTo***, nastavit konkrétní hodnotu průhlednosti. Hodnota průhlednosti se musí pohybovat v rozmezí od 0 do 255, kdy 0 představuje zcela průhledný objekt.

U objektů s rozhraním **IShape** se samotná průhlednost dá nastavit pouze změnou barvy výplně, která může mít nastavenou i průhlednost.

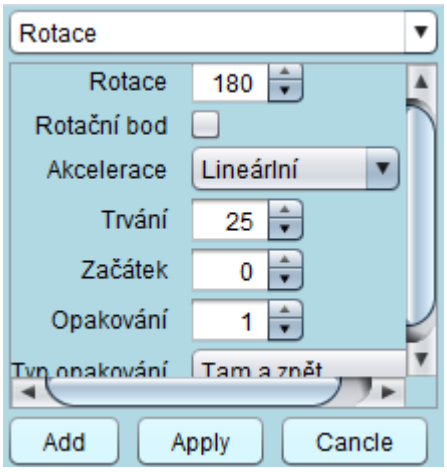
5 Implementace grafických komponent

Aplikace od své předchozí verze[6] změnila strukturu programu na vytvoření několika komponent pro rozdělení grafických panelů. Komponenty jsou v podstatě třídy, které pod sebou spravují další třídy, ale vystupují přes své rozhraní jako jeden funkční celek.

5.1 Třída AnimationProperty

Tato komponenta GUI obsahuje ovládací prvky k tvorbě, odstranění, aplikování změn a náhledu nastavení animací. Tato komponenta je umístěna v levém dolním sloupci hlavního okna aplikace. Panel obsahuje výsuvný seznam všech dostupných typů animací a tlačítka k přidání a odstranění animace, tlačítko k potvrzení nastavených hodnot a tlačítko k zrušení provedených změn.

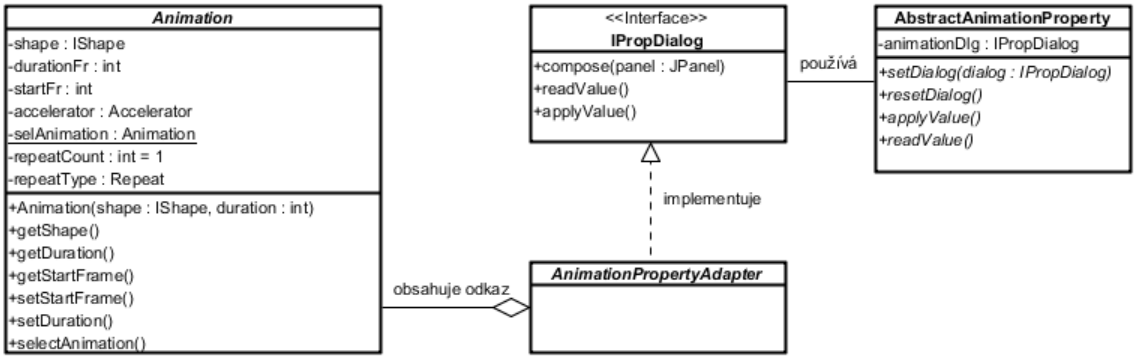
Třída, která tuto komponentu reprezentuje, se nazývá **AnimationProperty** a dědí ze třídy **AbstractAnimationProperty**. Z abstraktní třídy předka implementuje metodu ***setDialog***, příjímající jako vstupní parametr třídu, která implementuje rozhraní **IPropDialog** a která představuje zobrazení vizuální podoby dialogového panelu konkrétní animace. Voláním funkce ***readValue*** se zaktualizují hodnoty v ovládacích prvcích dialogového panelu podle hodnot animace a naopak metodou ***applyValue*** se podle těchto metod nastaví vlastnosti animace.



Obrázek 5 .1: *Animační modifikační panel*

Zobrazení animačních parametrů

Pro řízení změn vlastností animací bylo třeba do dialogového panelu přidávat ovládací prvky pro změnu hodnot. Tyto ovládací prvky ale závisí na konkrétním typu animace, takže bylo nutné svázat třídu **AnimationProperty**, se třídami odvozených od třídy **Animation**. Tyto dvě třídy ale spolu vzájemně nemusí souviset, takže pro řešení problému sáhnul po návrhovém vzoru Adapter[4].



Obrázek 5 .2: *Zobrazení parametrů animací s využitím návrhového vzoru Adapter*

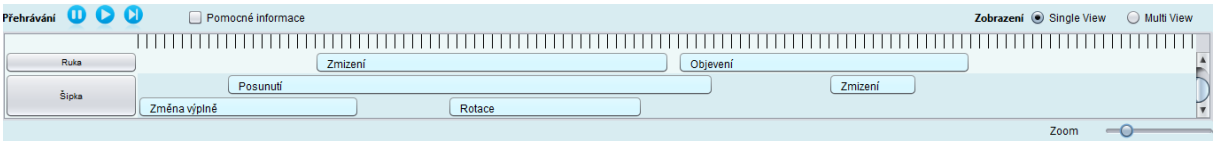
Rozhraní **IPropDialog** vystupuje jako rozhraní adaptéru pro dialogové okno, které musí umět reprezentovat libovolný objekt v podobě vykreslení ovládacích prvků, umožňující jej měnit. Metodou ***compose*** nasází komponenty do odkazovaného **JPanelu**, metodou ***readValue*** aktualizuje hodnoty ovládacích prvků podle aktuálního stavu modifikovaného objektu a metodou ***applyValue*** upraví hodnoty modifikovaného objektu podle hodnot v ovládacích prvcích.

5.2 Třída AnimationTimeLine

Komponenta GUI, která obsahuje ovládací prvky pro přehledné znázornění jednotlivých animovaných objektů a jeho animací v časovém uspořádání, jaké známe z jiných animačních nástrojů. Uživateli by hned na první pohled mělo být jasné, jaké animace jsou u objektů použity a kdy se provedou.

Třída, která tuto komponentu realizuje, se nazývá **AnimationTimeLine** a dědí z abstraktní třídy **AbstractAnimationTimeLine**. Tuto třídu nalezneme v balíku *dialog.dlgMain.gui.animationtimeline*. Třída nabízí funkčnost pro přecházení do konkrétního času, výběr animace a zobrazení nebo skrytí pomocných animačních informací. Pro pohodlnější zobrazení animací na časové ose byl také přidán ovládací prvek v pravém dolním rohu, který časovou osu zvětšuje nebo zmenšuje.

Pro možnost řízení náhledu animací jsou zde ovládací prvky v podobě tlačítek pro spuštění, zastavení a tlačítka zpětného převinutí na začátek scény. Všechna tato tlačítka obsluhují reakci na kliknutí myši přes vlastní třídu **ClickListenerTimeLine**, která dědí z třídy **MouseAdapter**, pro reakci na události myši. Při každém odchycení takovéto události, se přes tovární třídu **ActionFactoryTimeLine** a její metodu ***getAction***, získá a vykoná akce, se kterou je tlačítko svázáno.



Obrázek 5 .3: *Komponenta časové osy*

Časová osa s prvky umístěnými v ní je pro složitost, tvořena vlastní třídou, která je pojmenována **AnimationTimeLineComponent**. Tato třída je umístěna v balíku *dialog.dlgMain.gui.animtimeline.components*, kde jsou kromě této třídy také další třídy vnitřních prvků této třídy. Třída **AnimatedShape**, která dědí z vizuální komponenty jazyka Java pro tlačítko (**JButton**) a která reprezentuje animovaný objekt, třída **AnimationItem**, která je také odvozena od vizuální komponenty tlačítka a ta reprezentuje samotnou animaci. Třída **AnimationRuler** je vizuální komponenta pravítka pro časovou osu, registrující vlastní posluchač reakce stisku tlačítka myši nad touto komponentou a změnou proměnné, uchovávající číslo vybraného snímku.

6 Popis funkčnosti

Představme si nyní, jak jsou vykonávány základní funkce aplikace a které třídy při tom každá operace potřebuje. Při implementaci funkčnosti jsem se opíral o fungující návrhové vzory, které v aplikaci zůstaly z předchozí verze[6].

6.1 Načtení dostupných animačních funkcí

Aby mohla být aplikace v budoucnu dále obohacena o další animační funkce, byl proces načtení všech animací implementován s pomocí prostředků reflexe jazyka Java a dynamického načtení objektů tříd.

Při startu programu se třídou **AnimationFactory** projde obsah balíku *dialog.animation.function* a získají se všechny nalezené třídy, dědící z abstraktní třídy **Animation**. Z animačních tříd, se voláním metody ***createAnimation***, získá její instance, která se uloží do kolekce s načtenými animacemi.

Tento proces je vykoná pouze jednou a pouze na začátku programu. Po celou dobu běhu aplikace se kolekce animací uchovává a lze ji získat voláním metody ***getAnimations***, kterou najdeme ve třídě **AnimationFactory**.

Způsob přidání nové animační funkce do programu je popsán v kapitole Možná rozšíření 7.1.

6.2 Vytvoření animace

Vytvoření animace má na starost třída **AnimationFactory**. Tvorba nové animace vychází z návrhového vzoru **Prototype** [5]. Samotná třída reprezentuje prototypovou kolekci všech typů animací a nové animace tvoří na základě jejich klonování.

Proces tvorby animace začíná už při výběru položky ze seznamu animací, kterou najdeme v levém sloupci aplikačního okna v komponentě **AnimationProperty**. Po výběru animace se vykoná akce implementována ve třídě **ActionSelectedAnimationToCreate**. Metoda ***execute*** této třídy zjistí, zda je vybrán nějaký objekt k animování, zda je pouze jeden a také neběží-li žádná animace. Jsou-li splněny podmínky, získá se instance vybrané animace a zavolá se metoda ***createAnimation*** třídy **AnimationFactory**, která nám vrátí novou instanci z prototypu. Nově vytvořená animace se pak označí metodou ***select*** třídy **AnimationController** jako vybrána. V poslední fázi se z animace získá voláním metody ***getPropertyDialog*** odkaz na třídu, která implementuje rozhraní pro vykreslení ovládacích prvků ke změně parametrů animace. Získaná třída se použije k nastavení dialogového panelu animace v komponentě **AnimationProperty** zavoláním metody ***setDialog***.

Akce vytvoření animace se ukončí převedením aplikace do stavu Vytvoření animace, kterou reprezentuje třída **StatecCeatingAnimation**.

6.3 Přidání animace

Proces přidání animace do třídy **AnimationController** začíná stiskem tlačítka „Přidat“ v modifikačním dialogu animace. Reakci na stisknutí tlačítka má zaregistrovanou přes metodu ***actionPerformed*** třída **AnimationProperty**, která tlačítko sama obsahuje. Třída **AnimationProperty**

využívá vlastní továrnu na akce (**ActionFactoryAnimationProperty**), kterou metodou ***getAction***, žádá o navrácení instance potomka třídy **AbstractAction**, k vykonání množiny příkazů, které celou aplikaci nastaví do správného stavu.

Továrna akcí nám v tomto případě vrátí instanci třídy **ActionAddAnimation**, kterou nalezneme v balíku *dialog.dlgMain.action.dlgAction*. Když je metodou ***execute*** tato akce vykonána, aplikují se nejprve všechny nastavené hodnoty z modifikačního dialogu animace, a až poté je vybraná animace zaregistrována v **AnimationController**. K získání animace, kterou zrovna přidáváme, slouží metoda ***getSelectedAnimation*** třídy **AnimationController**. Předtím je totiž animace vždy vybrána metodou ***selectAnimation***, a to když si vybereme typ animace ze seznamu animací.

Pro registrování animace do třídy **AnimationController** se nejprve uloží záloha animovaného objektu do kolekce všech záloh a upraví se maximální délka celé animační scény tak, aby se přidávaná animace vykonala celá. Po těchto krocích se i samotná instance animace uloží do kolekce animací a nastaví se aktuální snímek na 0.

V závěru se voláním metody ***setDialog*** s parametrem *null* vymažou všechny komponenty v modifikačním dialogu **AnimationProperty** a přes třídu **StateHolder** se inicializuje aplikace do stavu modifikace objektů.

6.4 Odstranění animace

Odstranění animace ze scény začíná, stejně jako u přidání animace, stiskem tlačítka v modifikačním panelu třídy **AnimationProperty**. Toto tlačítko, pojmenováno „Odstranit“, volá obsluhu události v metodě ***notifyChange***, odkud se získá továrna na obsluhované akce (**ActionFactoryAnimationProperty**) a přes metodu ***getAction*** se získá instance třídy **ActionDeleteAnimation**. Třída má za úkol nastavení aplikace do stavu s odstraněním vybrané animace a aktualizaci všech komponent, které na absenci animace reagují.

Obsluha události odstranění animace ze scény provádí odebrání registrované animace ze třídy **AnimationController** metodou ***remove*** a to na základě toho, že získá vybranou animace metodou ***selectAnimation***.Ve třídě **AnimationController** se kromě odstranění reference na animaci, kontroluje, zda je možné uvolnit i zálohu animovaného objektu. To znamená, zda už žádná jiná instance animace neodkazuje na stejný objekt.

Po odebrání všech objektů, se metodou ***resetDialog*** zruší všechny komponenty, pro nastavení animace a metodou ***refresh***, nad třídou **AnimationTimeLine**, se překreslí časová osa. Celý proces je pak ukončen nastavením celé aplikace do stavu pro modifikaci objektu.

6.5 Uložení a načtení animací

Za uložení animace do souboru je zodpovědná akce třídy **ActionSaveAnimation**, která je umístěna v balíku *dialog.dlgMain.action.actionDlg*. Při vykonání této akce se metodou ***clearHistory*** nad třídou **AnimatableCanvas** vyprázdní historie a aplikace se převede do stavu při spuštění aplikace. Následně se voláním ***showSaveDialog*** nad instancí třídy **JFileChooser**, zobrazí dialogové okno pro výběr umístění pro uložení souboru projektu. Posledním krokem je uložení dat pro projekt do soubor. Tuto operaci realizuje použitá knihovna **XStream**, která ukládá do XML objekt třídy **SaveObjects**, ve

kterém jsou uloženy všechny objekty registrované ve třídě **AnimatableCanvas** a všechny objekty animací ze třídy **AnimationController**.

Načtení projektu probíhá obdobně jako uložení. Z vybraného souboru se načtou v podobě XML informace o uložených objektech, ze kterých knihovna **XStream** vytvoří objekt třídy **SaveObject** a v nich instance objektů a animací. Objekty jsou jeden po druhém, metodou ***addShape***, postupně vkládány do třídy **AnimatableCanvas**. Animace jsou zase ukládány do třídy **AnimationController**.

6.6 Export animace do video souboru

Aplikace za pomocí knihovny **Xuggler**[1] umí také vytvářet video soubor z jednotlivých snímků z celé animační scény. Celý proces je odstartován vykonáním akce třídy **ActionCreateAnimationMovie**. Prostřednictvím instance aplikace si voláním metody získáme instanci třídy **AnimationEncder**, která v metodě encode řídí tvorbu videa.

Než začneme vkládat jednotlivé snímky scény, vytvoříme si proud video souboru. Přes **ToolFactory** a statickou metodu ***makeWriter*** definujeme umístění proudu. Metodou ***addVideoStream*** nastavíme zvolený typ kodeku a rozlišení v pixelech. Standardní nastavení je v poměru 1200 na 800 pixelů s kodekem MPEG-4. Nyní už se v cyklu opakuje volání metody ***setActualFrame*** třídy **AnimationController** k nastavení scény v tomto snímku. K získání obrázku aktuální scény zavoláme metodu ***paintFrame***, která vykreslí aktuální pozici objektů a vrací instanci třídy **BufferedImage**. Tento obrázek se vloží metodou ***encodeVideo*** do proudu s nastaveným časem trvání snímku. Po vložení všech snímků se video proud uzavře metodou ***close***.

7 Možnosti rozšíření

Název kapitoly Možnosti rozšíření je myšlen jako nasměrování k nedostatkům, které aplikace má a k definování možného řešení, které tento problém řeší. Jsou to zejména věci týkající se vložení nové funkcionality nebo rozšíření stávající.

7.1 Nové animační funkce

Pokud si už nevystačíme s nabízenými animacemi a budeme chtít přidat novou funkci, stačí vytvořit novou třídu, která bude dědit z abstraktní třídy pro všechny animace **Animation**. Získáme tak všechny členské proměnné této třídy.

První, co musíme udělat, je implementovat konstruktor nové třídy, ve kterém se budou inicializovat nově vložené členské proměnné, které definují vlastnosti nové animační funkce. Příkladem může být následující kód pro konstruktor bézierové křivky.

```
public BezierCurveAnimation(IShape shapes, int duration) {
      super(shapes, duration);
      startPt = new Point2D.Double(0, 0);
      ctrlPt1 = new Point2D.Double(0, 150);
      ctrlPt2 = new Point2D.Double(200, 150);
      endPt = new Point2D.Double(200, 0);
}
```

Inicializace proměnných je důležitá, protože po načtení seznamu animací třídou **AnimationFactory**, dochází pak už jen ke klonování instance třídy, a předpokládá se, že tato instance už bude mít implicitně nastavené hodnoty. Těchto implicitních hodnot se využívá k zobrazení náhledu animace, při jejím výběru.

Další důležitou součástí implementace je překrytí metody ***clone***, která je volána třídou **AnimationFactory**, při vytváření animací podle vybraného prototypu animace. Metoda ***clone*** je už podle svého názvu metodou, která by měla vrátit novou instanci animace s překopírováním všech proměnných definující animaci.

Samotný výpočet stavu animovaného objektu v čase je v metodě ***refresh***. Jediný parametr této funkce představuje časovou reprezentaci stavu animace. Pokud je tento parametr kupříkladu nastaven na hodnotu 0.5, znamená to, že má animace nastavit objekt do stavu polovičního časového zpracování.

```
public void refresh(double t) {
      double xpos = move_x * t;
      double ypos = move_y * t;

      AffineTransform transform = getTransform();
      transform.setToIdentity();
      transform.translate(xpos, ypos);
}
```

Aby se zobrazilo správně jméno animace, je třeba překrýt metodu ***toString*** a ***getAnimationName***, která v naší verzi vrací řetězec názvu typu animace.

Metodou ***paintInfo*** zase docílíme vykreslení pomocných informací, které na první pohled dávají představu o tom, co se bude s objektem dít. Můžeme třeba vykreslit trajektorii pohybu objektu nebo náhled výsledného stavu objektu.

7.2 Změna modifikačních panelů animací

Každá animační třída by měla implementovat funkci ***getPropertyDialog***, která vrací odpovídající třídu modifikačního panelu animace. Třída pro dialogová okna musí implementovat rozhraní **IPropDialog**. Modifikační panel představuje zděděnou třídu **JPanel** knihovny swing, do které se vkládají další komponenty ovládacích prvků jako je **JCombobox**, **JTextField**, **JLabel**, **JSpinner**. Vždy podle toho, jakými hodnotami potřebujeme nastavit funkci.

Z důvodu efektivního přístupu k paměťovým prostředkům by měla být třída modifikačního dialogu animace tvořena s návrhovým vzorem Jedináček[5] (**Singleton**). Stačí totiž vždy jeden modifikační panel pro všechny animace stejného typu. Ušetří se tak i čas pro vyhrazení paměťového místa pro další instanci a není potřeba, aby GC Javy čistil místo po uvolněných prostředcích.

Za načtením hodnot z animační funkce stojí metoda ***readValue*** a za uložení hodnot z modifikačního panelu zpět do instance animace metoda ***applyValue***.

7.3 Rozšíření funkčnosti časové osy

Jako možnou další funkčnost, kterou je vhodné rozšířit je přesouvání animací na časové ose do jiných spouštěcích časů jednoduchým přetažením myši. Z hlediska možné nepřehlednosti by bylo také vhodné doimplementovat časovou osu s pravítkem souřadnic každého snímku.

7.4 Obohacení tvorby video animace

Dosavadní stav aplikace umožňuje pouze tvorbu animace v podobě video souboru s použitím jednoho druhu kodeku a bez nastavení poměru stran, snímkovací frekvence nebo třeba kvality videa. Pro pozdější rozšíření by tak nebylo od věci umožnit bližší nastavení parametrů videa.

8 Závěr

V této práci jsem rozšířil funkčnost předešlého návrhu aplikace, která vycházela z diplomové práce Ing. Karla Šuty, aby mohla být aplikace využitá k jednoduchému návrhu animací a k prezentaci při výuce. Program byl přepracován do takové podoby, že nyní lze objektům definovat animaci otáčení, změnu měřítka, objevení, zmizení, přesunutí, změnu barvy a pohyb po křivce a k tomu nastavit čas a délku spuštění. Aplikace si také umí poradit s nastavením nestejnoměrného pohybu, způsobu a počtu opakování animace. Pro rychlejší a intuitivnějším přístupu k možnostem aplikace, bylo přepracováno uživatelské rozhraní a přidána komponenta časové osy s animacemi.

Ze způsobu implementace aplikace, nabízí vypracované řešení možnost dalšího vylepšení vzhledu a funkčnosti. S ohledem na další rozšíření, jsem dbal na návrh v podobě návrhových vzorů a zachoval už existující návrhové vzory. Aby bylo případné rozšíření aplikace dobře implementovatelné, je zdrojový kód aplikace zdokumentován prostředky jazyka Java.

Tato práce mi přinesla obohacení mých znalostí v rámci programování v jazyce Java a přinesla mi zkušenost při práci v týmu. Také jsem si mohl vyzkoušet, jak důležitý je k pochopení existující struktury programu, správný a přehledný návrh předešlé verze aplikace, který zůstal zachován. Součástí práce je funkční aplikace, která by měla splňovat všechny zadané požadavky.

Použitá literatura

[1] **XUGGLE INC.** *Xuggler* [počítačový soubor] Ver. 5.4. Dostupné z: http://www.xuggle.com/xuggler. Java jar library to uncompress, re-compress media file

[2] **VISUAL PARADIGM INTERNATIONAL**. *Visual Paradigm for UML* [počítačový soubor]. Ver. 10.1. 2013. Dostupné z: http://www.visual-paradigm.com/product/vpuml/. UML modeling software that support UML, SysMl, ERD, use case and diabase design.

[3] **THE TORTOISE SVN TEAM**. *TortoiseSVN.* [počítačový soubor]. Ver. 1.7.11. 2012. Dostupné z: http://tortoisesvn.net/. Apache Subversion (SVN) klient, implemented as a window shell extension.

[4] **PECINOVSKÝ**, Rudolf. *Návrhové vzory*. Vyd. 1. Brno: Computer Press, 2007, 527 s. ISBN 978-80-251-1582-4

[5] **GAMMA**, Erich. *Návrh programů pomocí vzorů*. Vyd. 1. Praha: Grada, 2003, 386 s. ISBN 80-247-0302-5.

[6] **ŠUTA,** Karel. Vektorový animátor. Ostrava, 2009. 39 s. Diplomová práce na Fakultě elektrotechniky a informatiky Vysoké školy báňské. Vedoucí diplomové práce Ing. David Ježek, Ph.D.

Seznam příloh

Součástí BP je CD s aplikací a uloženými scénami a objekty.

Adresářová struktura přiloženého CD/DVD:

Adresář	Obsah
/Text/	Text bakalářské práce
/Prirucka/	Uživatelská dokumentace
/Aplikace/	Zdrojový kód aplikace
/Ukazka/	Předpřipravené animační scény

Uživatelská dokumentace

Zpracoval: Martin Gold, Zděnek Gold

Úvod

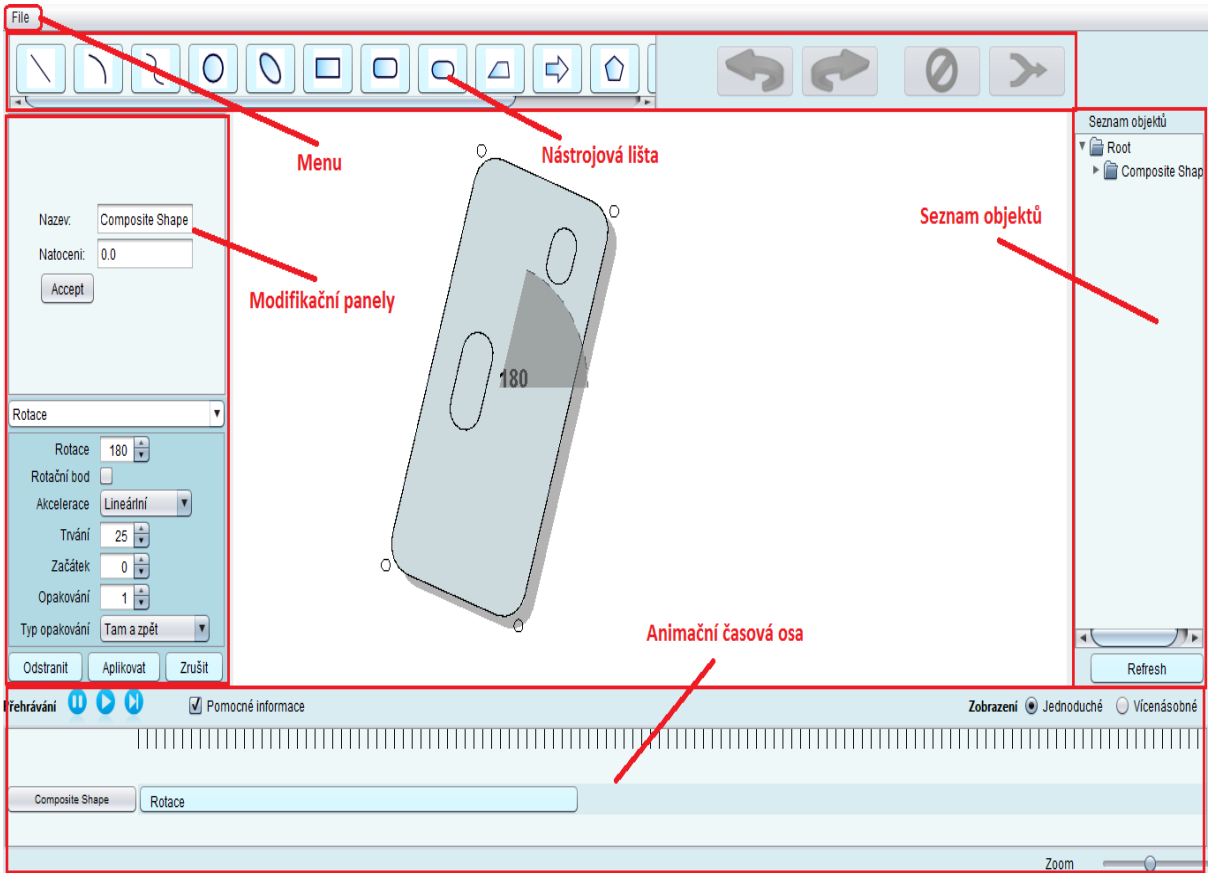
Tento grafický animátor byl vytvořen jako bakalářská práce Martina Golda a Zdeňka Golda. Jde o jednoduchou aplikaci, která by měla být vhodná pro tvorbu materiálů pro výukové účely.

Program umožňuje vytváření objektů za pomoci již nadefinovaných tvarů včetně možnosti jejich následného seskupování do větších celků. Ke každému objektu zde existuje i výčet parametrů, pomocí kterých lze tvary editovat.

Ke vytvořenému objektu lze následně přiřadit jednu nebo více předdefinovaných animací, jejichž nastavení lze taktéž ovlivňovat.

Výslednou animaci lze nakonec vyexportovat do video souboru typu MPEG4.

Grafické uživatelské rozhraní aplikace



Menu

Hlavní panel obsahuje přístup k možnostem týkajících se celého projektu. Obsaženo je zde jen jedno tlačítko, nazvané Soubor, pod kterým se nacházejí tyto možnosti:

- Uložení animace – umožňuje uložení celého projektu včetně vytvořených animací
- Načtení animace – umožňuje načíst již dříve vytvořený projekt
- Uložení tvaru – umožňuje uložit nadefinovaný tvar
- Načtení tvaru – umožňuje načíst již dříve vytvořený tvar
- Vytvoř video – umožňuje vyexportovat vypracovanou animaci do spustitelného video souboru MPEG4

Nástrojová lišta

Nástrojová lišta tvoří jednu z nejpodstatnějších částí aplikace. Jsou zde umístěny jak základní tlačítka pro ovládání programu, tak i seznam objektů, které lze vytvářet. Mezi základní tlačítka pro ovládání patří možnost postoupit dopředu/zpět, které umožňují posunovat dále/zpět stav pomocí záchytných bodů uložených v historii změn. Dále se zde nachází ještě tlačítko pro odstranění objektu

(pozn. u sloučených objektů slouží pouze k odstranění celkové vazby a zachovává tak všechny členské objekty) a tlačítko pro sloučení více objektů do jedné skupiny. Nakonec je zde ještě umístěn seznam tlačítek, pomocí kterých lze vytvářet všechny předdefinované tvary.

Seznam obsahuje (zleva): úsečku (line), Bézierovu kvadratickou křivku (Bezier quadratic curve), Bézierovu kubickou křivku (Bézier cubic curve), kruh (circle), elipsu (ellipse), obdélník (rectangle), obdélník s kulatými rohy (rounded rectangle), ovál (oval), lichoběžník (trapezoid), šipku (arow), n-úhelník (ngon), hvězdu (star), kvádr (block) a textové pole (text box).

Modifikační panely

Modifikační panely se nacházejí na levém okraji aplikace a umožnují uživateli editovat jak strukturu objektu, tak i průběh jeho animace.

Editace objektu

V této části aplikace jsou uživateli nabídnuty veškeré parametry, pomocí kterých lze upravovat celkový vzhled právě označeného objektu. Tyto parametry nabízí jak základní modifikace, které se týkají například výšky a šířky tvaru, tak i pokročilejší funkce. Mezi ně patří například možnost zapnout či vypnout stín, měnit odsazení stínu, měnit barvy výplně a ohraničení, měnit styl ohraničení nebo například modifikovat celý objekt pomocí transformace zkosením. Seznam těchto funkcí však zde nemůže být vypsán uceleně, poněvadž výčet aktuálních funkcí záleží na právě označeném objektu. Některé tvary totiž nabízí jen malý výběr funkcí a naopak některé tvary nabízí ještě navíc některé speciální vlastnosti charakteristické pouze pro ně. Příkladem může být textové pole, které kromě běžných funkcí nabízí uživateli ještě široké možnosti práce s textem.

Editace animací

V panelu pro editaci animací se uživateli nabízí prostředky k definování vlastností animací nebo k jejich přidávání a odebírání. Součástí tohoto panelu je i vysouvací seznam s nabídkou dostupných typů animací. U každého typu animace se stanovují společné parametry: snímek počátku animace, délka animace a akcelerátor. Některé speciální animace, kromě těchto základních vlastností využívají hodnoty v podobě vektoru, rotačního bodu apod. Také existují možnosti nastavit počet a typ opakování. V dolní části panelu jsou tři tlačítka:

- Přidat/Odstranit - přidává/odstraňuje aktuálně definovanou animaci do/ze scény
- Aplikovat - potvrzuje nastavené hodnoty nastavené v modifikačním panelu a aplikuje je na animaci
- Zrušit - zahazuje současně nastavené hodnoty aplikované na animaci

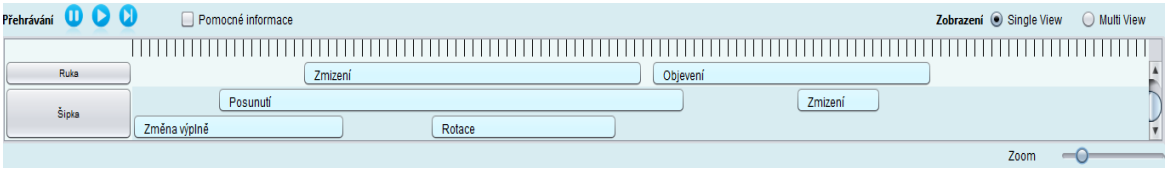
Seznam objektů

Tento panel je umístěn na pravém okraji aplikace a slouží k snadnější orientaci v programu. Hlavní částí je okno, ve kterém se vypisuje seznam jmen všech použitých objektů v tomto projektu. Seznam však uživateli nenabízí prostý výčet jmen, ale dbá také na zachovávání jejich hierarchické struktury. Ta je využívána hlavně při shlukování objektů do skupin. V tomto případě se v seznamu vytvoří adresář reprezentující takto sloučenou skupinu a do něj jsou přesunuty všechny členské

objekty. S tímto seznamem je pak spojeno tlačítko Aktualizovat, které po jeho stisknutí přepracuje celý zobrazený seznam.

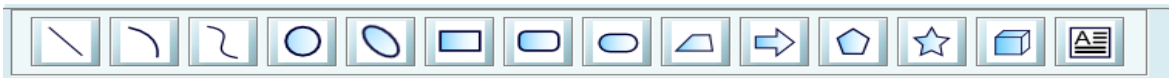
Animační časová osa

V dolní části okna aplikace je umístěna komponenta pro výběr animací a nastavování aktuálního snímku scény. V levém horním rohu tohoto panelu jsou tlačítka k ovládání náhledu animování objektů. Zaškrtávací tlačítko povolí nebo zakáže vykreslování informací o animacích přímo na plátně. V pravém dolním rohu panelu je posuvný jezdec umožňující přibližování/oddalování časové osy.



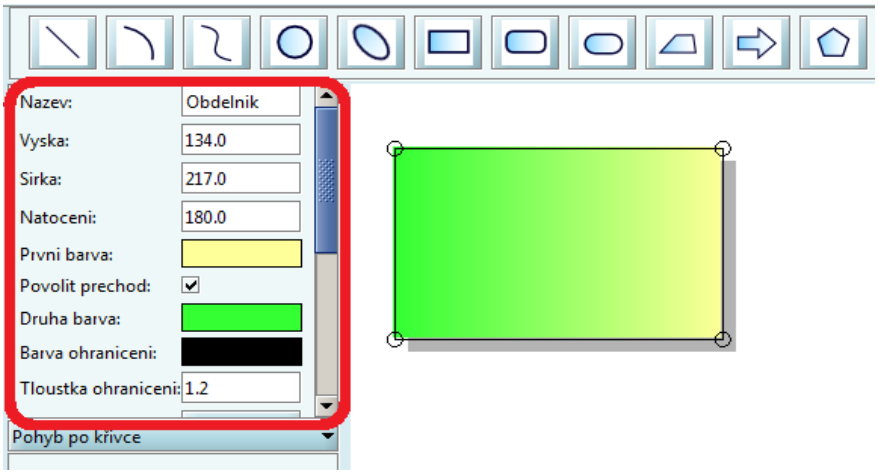
Vytváření a editace objektů

Vytváření jednotlivých objektů je velmi jednoduché. Nejdříve si uživatel vybere jeden z předdefinovaných tvarů nacházejících se v seznamu objektů. Vybraný objekt poté vytvoříme kliknutím na kreslící plátno a následným tahem myši. Pro přehlednější vytváření tvarů je uživateli průběžně během tahu zobrazována aktuální velikost objektu.

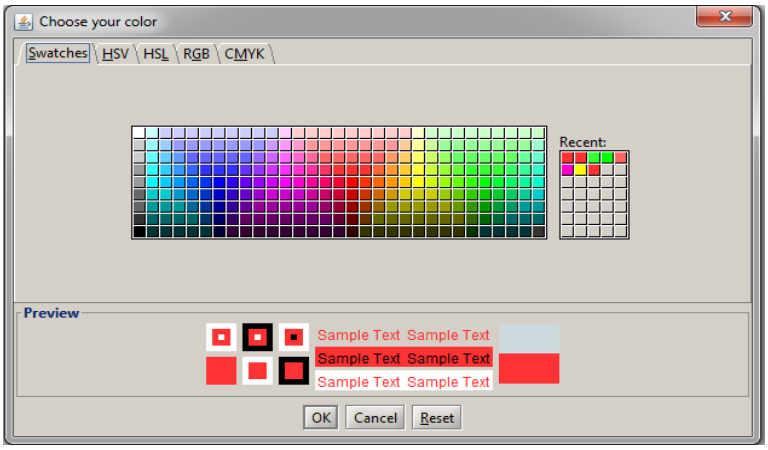


Pro označování objektů je možno použít jak kliknutí na daný tvar, tak možnost obdélníkového výběru, pomocí něhož lze vybrat i více objektů najednou. Označenému objektu se následně pro lepší přehlednost zvýrazní všechny jeho řídící body.

Kliknutím a následným tahem lze takto označený objekt přesouvat. Objekt však můžeme i otočit. To provedeme tak, že označíme jeden z řídících bodů tvaru, ten bude představovat střed otáčení. Poté opět využijeme funkce kliku a tahu, pomocí kterého určíme úhel otočení tohoto objektu.



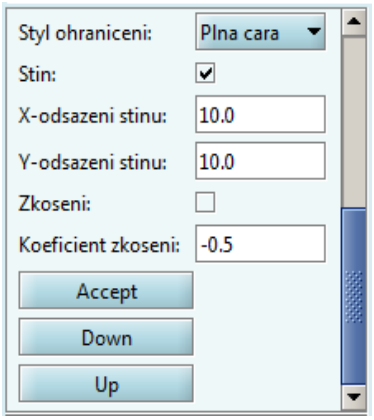
Po označení objektu lze také pozorovat překreslení modifikačního panelu, nacházejícího se na levé straně aplikace. Jestliže označíme více než jeden objekt nebo naopak žádný, zobrazí se uživateli prázdný dialog. Při označení právě jednoho objektu se však v tomto panelu zobrazí výčet všech parametrů, které můžou být u tohoto typu tvaru měněny.



V tomto okně pak může uživatel měnit všechny dostupné parametry a to buď přepsáním jejich hodnoty, zaškrtnutím nabízené možnosti nebo jednoduchým vybráním jedné možnosti z předpřipravené nabídky. Mimo to se zde vyskytují i atribut pro změnu barvy. Jejich změna se provádí klinutím na pole s barvou. Po této akci se uživateli zobrazí nové okno, které nabízí širokou škálu možností pro nadefinování požadované barvy. Může být využito například barevné schéma RGB, HSV, HSL nebo CMYK, kde lze nadefinovat i průhlednost určené barvy. Kromě těchto barevných schémat však může být využita i paleta s předdefinovanými barvami.

Změny, které nijak neovlivňují geometrii objektu (jako je například změna barvy) se projeví okamžitě. U ostatních změn je třeba potvrdit jejich nové hodnoty pomocí potvrzovacího tlačítka, které

lze vidět na obrázku níže. Povšimnout si zde můžeme i dalších dvou tlačítek, při jejichž stisknutí se označený objekt přesune buďto více do pozadí nebo naopak více do popředí.

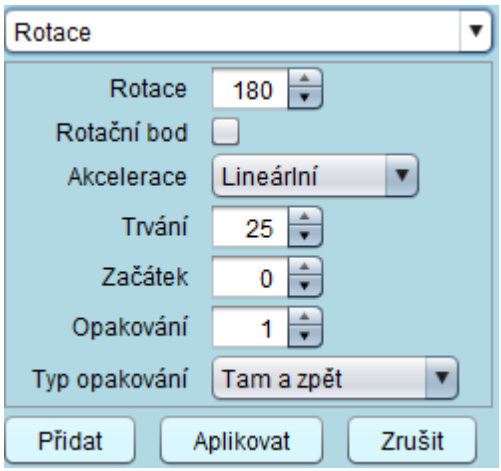


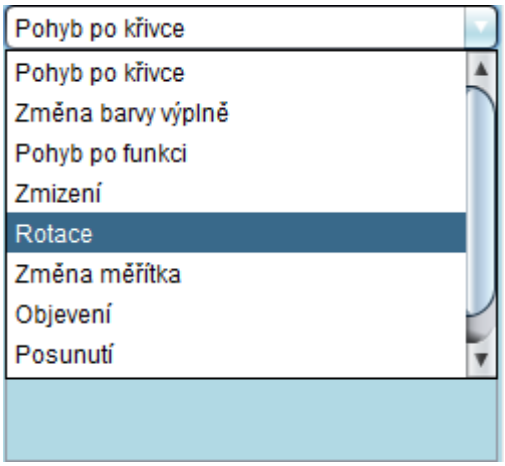
Přiřazování a editace animací

Chceme-li s vytvořenými objekty vytvořit animovanou scénu, je třeba zvolený objekt nejprve označit. Po výběru objektu se přesuneme do modifikačního panelu a z vysouvací nabídky si vybereme typ animace. V základní verzi programu máme k dispozici tyto typy animací:

- Pohyb po křivce - objekt opisuje trajektorii kubické bézierové křivky
- Změna barvy výplně - objekt plynulým přechodem mění barvu výplně na zvolenou barvu
- Pohyb po funkci - objekt opisuje trajektorii definovanou nějakou funkcí
- Zmizení - objekt mění svoji průhlednost do ztracena
- Rotace - objekt se otáčí kolem svého středu nebo kolem bodu mimo střed
- Změna měřítka - objekt plynule mění výšku a šířku podle násobku těchto velikostí
- Objevení - objekt mění svoji průhlednost až do plného zobrazení
- Posunutí - objekt se přesune o stanovený vektor posunutí
- Animace průhlednosti - objekt mění svoji průhlednost do nastavené hodnoty průhlednosti

Po zvolení typu animace se v editačním panelu zobrazí její parametry. U každého objektu se nastavují různé parametry. Viz obrázek níže.

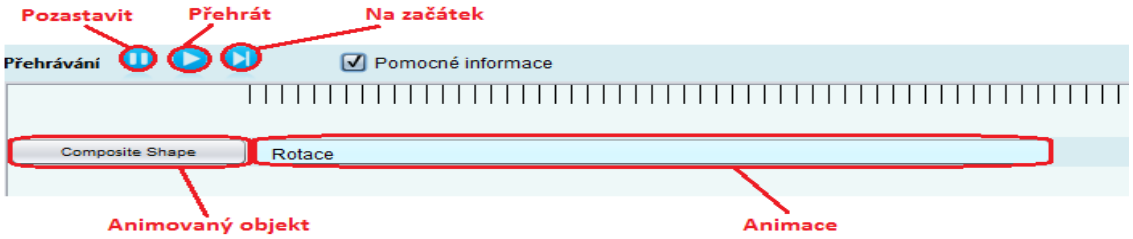




Nyní je na nás, jak nastavíme řídící hodnoty animace. U každé animace budou vždy vlastnosti:

- Akcelerace - nastavuje zrychlení animace
- Trvání - nastavuje délku animace ve snímcích
- Začátek - snímek, ve kterém animace začíná
- Opakování - počet opakovacích cyklů
- Typ opakování - k dispozici jsou dvě možnosti (Tam a zpět - po dokončení jednoho cyklu se změní směr a animuje se pozpátku, Vždy od začátku - po dokončení cyklu se animace vrací do počátečního stavu a další cyklus probíhá od začátku)

Po nastavení všech hodnot, přidáme tlačítkem “Přidat” animaci do scény. Po přidání se animace objeví v časové ose. Tlačítkem “Zrušit” zahodíme všechny změny a animaci nepřidáme.



Nyní, když už je animace přidána do scény, můžeme si ji přehrát. K tomu slouží tlačítko na časové ose. Pokud zjistíme, že animace nám nevyhovuje, stačí na ní tlačítkem myši kliknout. Aktuální nastavení vybrané animace se nám znovu zobrazí v editačním panelu. Po změně hodnot stačí stisknout tlačítko “Aplikovat” a animace si uloží nové nastavení.

Pokud chceme animaci ze scény odebrat, musíme ji nejprve vybrat z časové osy a poté tlačítkem “Odstranit” ji v editačním panelu odstraníme.